Kauno technologijos universitetas

Informatikos fakultetas

HashMap

P175B014 Duomenų struktūrų trečias laboratorinis darbas

Projekto autorius
Gustas Klevinskas

Akademinė grupė
IFF-8/7

Kaunas, 2019

Turinys

HashMap metodai

containsValue(K key)

```
public boolean containsValue(Object value) {
    return toString().contains(value.toString());
}
```

```
======== containsValue testas ========
========Pradinis sąrašas=======
  [ 0 ]
  [ 1 ]    -->Antanas Škėma - Balta drobulė (1958) ☆6.9
  [ 2 ]
  [ 3 ]
  [ 4 ]
  [ 5 ]    -->Mark Twain - Moby Dick (2019) ☆0.0
  [ 6 ]
  [ 7 ]
  [ 8 ]
  [ 9 ]
 [ 10 ]
 [ 11 ]    -->Maironis - Metai (1894) ☆9.9
 [ 12 ]
 [ 13 ]
 [ 14 ]    -->Balys Sruoga - Kupstas (1980) ☆0.0
 [ 15 ]
****** Bendras porų kiekis yra 4
======== Atvaizdžio pabaiga =======
Mark Twain - Moby Dick (2019) ☆0.0 - true
Nauja knyga - Labai nauja (2019) ☆0.0 - false
```

putIfAbsent(K key, V value)

```java
public V putIfAbsent(K key, V value) {
    if (get(key) == null) {
        put(key, value);
        return null;
    }

    return value;
}
```

```
======== putIfAbsent testas ========
========Pradinis sąrašas=======
  [ 0 ]
  [ 1 ]    -->Antanas Škėma - Balta drobulė (1958) ☆6.9
  [ 2 ]
  [ 3 ]
  [ 4 ]
  [ 5 ]    -->Mark Twain - Moby Dick (2019) ☆0.0
  [ 6 ]
  [ 7 ]
  [ 8 ]
  [ 9 ]
 [ 10 ]
 [ 11 ]    -->Maironis - Metai (1894) ☆9.9
 [ 12 ]
 [ 13 ]
 [ 14 ]    -->Balys Sruoga - Kupstas (1980) ☆0.0
 [ 15 ]
****** Bendras porų kiekis yra 4
======== Atvaizdžio pabaiga =======
Mark Twain - Moby Dick (2019) ☆0.0 - Mark Twain - Moby Dick (2019) ☆0.0
Šefas - Kepimo knyga (2000) ☆0.0 - null
========Sąrašas po pridėjimo=======
  [ 0 ]
  [ 1 ]    -->Antanas Škėma - Balta drobulė (1958) ☆6.9
  [ 2 ]
  [ 3 ]
  [ 4 ]
  [ 5 ]    -->Mark Twain - Moby Dick (2019) ☆0.0
  [ 6 ]
  [ 7 ]    -->Šefas - Kepimo knyga (2000) ☆0.0
  [ 8 ]
  [ 9 ]
 [ 10 ]
 [ 11 ]    -->Maironis - Metai (1894) ☆9.9
 [ 12 ]
 [ 13 ]
 [ 14 ]    -->Balys Sruoga - Kupstas (1980) ☆0.0
 [ 15 ]
****** Bendras porų kiekis yra 5
======== Atvaizdžio pabaiga =======
```

replace(K key, V oldValue, V newValue)

```java
public boolean replace(K key, V oldValue, V newValue) {
    if (get(key) != null && get(key).equals(oldValue)) {
        Node<K, V> node = getInChain(key, table[hash(key, ht)]);
        node.value = newValue;
        return true;
    }

    return false;
}
```

```
======== replace testas ========
========Pradinis sąrašas=======
  [ 0 ]
  [ 1 ]    -->Antanas Škėma - Balta drobulė (1958) ☆6.9
  [ 2 ]
  [ 3 ]
  [ 4 ]
  [ 5 ]    -->Mark Twain - Moby Dick (2019) ☆0.0
  [ 6 ]
  [ 7 ]
  [ 8 ]
  [ 9 ]
 [ 10 ]
 [ 11 ]    -->Maironis - Metai (1894) ☆9.9
 [ 12 ]
 [ 13 ]
 [ 14 ]    -->Balys Sruoga - Kupstas (1980) ☆0.0
 [ 15 ]
****** Bendras porų kiekis yra 4
======== Atvaizdžio pabaiga =======
Atnaujintas pav - Kepimo knyga (2000) ☆0.0 keičiamas į Maironis - Metai (1894) ☆9.9
false
Maironis - Metai (1894) ☆9.9 keičiamas į Atnaujintas pav - Kepimo knyga (2000) ☆0.0
true
========Sąrašas po pakeitimo=======
  [ 0 ]
  [ 1 ]    -->Antanas Škėma - Balta drobulė (1958) ☆6.9
  [ 2 ]
  [ 3 ]
  [ 4 ]
  [ 5 ]    -->Mark Twain - Moby Dick (2019) ☆0.0
  [ 6 ]
  [ 7 ]
  [ 8 ]
  [ 9 ]
 [ 10 ]
 [ 11 ]    -->Atnaujintas pav - Kepimo knyga (2000) ☆0.0
 [ 12 ]
 [ 13 ]
 [ 14 ]    -->Balys Sruoga - Kupstas (1980) ☆0.0
 [ 15 ]
****** Bendras porų kiekis yra 4
======== Atvaizdžio pabaiga =======
```

replaceAll(V oldValue, V newValue)

```java
public void replaceAll(V oldValue, V newValue) {
    for (Node<K, V> i : table) {
        for (Node<K, V> n = i; n != null; n = n.next) {
            if (n.value.equals(oldValue))
                n.value = newValue;
        }
    }
}
```

```
======== replaceAll testas ========
========Pradinis sąrašas=======
  [ 0 ]
  [ 1 ]    -->Antanas Škėma - Balta drobulė (1958) ☆6.9
  [ 2 ]
  [ 3 ]
  [ 4 ]
  [ 5 ]    -->Mark Twain - Moby Dick (2019) ☆0.0
  [ 6 ]
  [ 7 ]
  [ 8 ]
  [ 9 ]
 [ 10 ]
 [ 11 ]    -->Maironis - Metai (1894) ☆9.9
 [ 12 ]    -->Mark Twain - Moby Dick (2019) ☆0.0
 [ 13 ]
 [ 14 ]    -->Balys Sruoga - Kupstas (1980) ☆0.0
 [ 15 ]
****** Bendras porų kiekis yra 5
======== Atvaizdžio pabaiga =======
========Sąrašas po pakeitimo=======
  [ 0 ]
  [ 1 ]    -->Antanas Škėma - Balta drobulė (1958) ☆6.9
  [ 2 ]
  [ 3 ]
  [ 4 ]
  [ 5 ]    -->Smagumynas - Duomenu strk. (1800) ☆0.0
  [ 6 ]
  [ 7 ]
  [ 8 ]
  [ 9 ]
 [ 10 ]
 [ 11 ]    -->Maironis - Metai (1894) ☆9.9
 [ 12 ]    -->Smagumynas - Duomenu strk. (1800) ☆0.0
 [ 13 ]
 [ 14 ]    -->Balys Sruoga - Kupstas (1980) ☆0.0
 [ 15 ]
****** Bendras porų kiekis yra 5
======== Atvaizdžio pabaiga =======
```

numberOfEmpties()

```java
public int numberOfEmpties() {
    int emptyCells = 0;

    for (Node<K, V> i : table) {
        if (i == null)
            emptyCells++;
    }

    return emptyCells;
}
```

```
======== numberOfEmpties testas ========
========Pradinis sąrašas=======
  [ 0 ]
  [ 1 ]    -->Antanas Škėma - Balta drobulė (1958) ☆6.9
  [ 2 ]
  [ 3 ]
  [ 4 ]
  [ 5 ]    -->Mark Twain - Moby Dick (2019) ☆0.0
  [ 6 ]
  [ 7 ]
  [ 8 ]
  [ 9 ]
 [ 10 ]
 [ 11 ]    -->Maironis - Metai (1894) ☆9.9
 [ 12 ]
 [ 13 ]
 [ 14 ]    -->Balys Sruoga - Kupstas (1980) ☆0.0
 [ 15 ]
****** Bendras porų kiekis yra 4
======== Atvaizdžio pabaiga =======
Tuščių celių - 12
```

HashMapOa

```
package edu.ktu.ds.lab3.klevinskas;

import edu.ktu.ds.lab3.utils.HashType;
import edu.ktu.ds.lab3.utils.Map;

import java.util.Arrays;

public class HashMapOa<K, V> implements Map<K, V> {

    private static final int DEFAULT_INITIAL_CAPACITY = 16;
    private static final float DEFAULT_LOAD_FACTOR = 0.75f;
    private static final HashType DEFAULT_HASH_TYPE = HashType.DIVISION;

    private Entry<K, V>[] table;
    private int size = 0;
    private int deletedEntries = 0;
    private float loadFactor;
    private HashType ht;

    public HashMapOa() {
        this(DEFAULT_HASH_TYPE);
    }

    public HashMapOa(HashType ht) {
        this(DEFAULT_INITIAL_CAPACITY, ht);
    }

    public HashMapOa(int initialCapacity, HashType ht) {
        this(initialCapacity, DEFAULT_LOAD_FACTOR, ht);
    }

    public HashMapOa(float loadFactor, HashType ht) {
        this(DEFAULT_INITIAL_CAPACITY, loadFactor, ht);
    }

    public HashMapOa(int initialCapacity, float loadFactor, HashType ht) {
        if (initialCapacity <= 0) {
            throw new IllegalArgumentException("Illegal initial capacity: " +
initialCapacity);
        }

        if ((loadFactor <= 0.0) || (loadFactor > 1.0)) {
            throw new IllegalArgumentException("Illegal load factor: " +
loadFactor);
        }

        this.table = new Entry[initialCapacity];
        this.loadFactor = loadFactor;
        this.ht = ht;
    }

    @Override
    public boolean isEmpty() {
        return size == 0;
    }
```

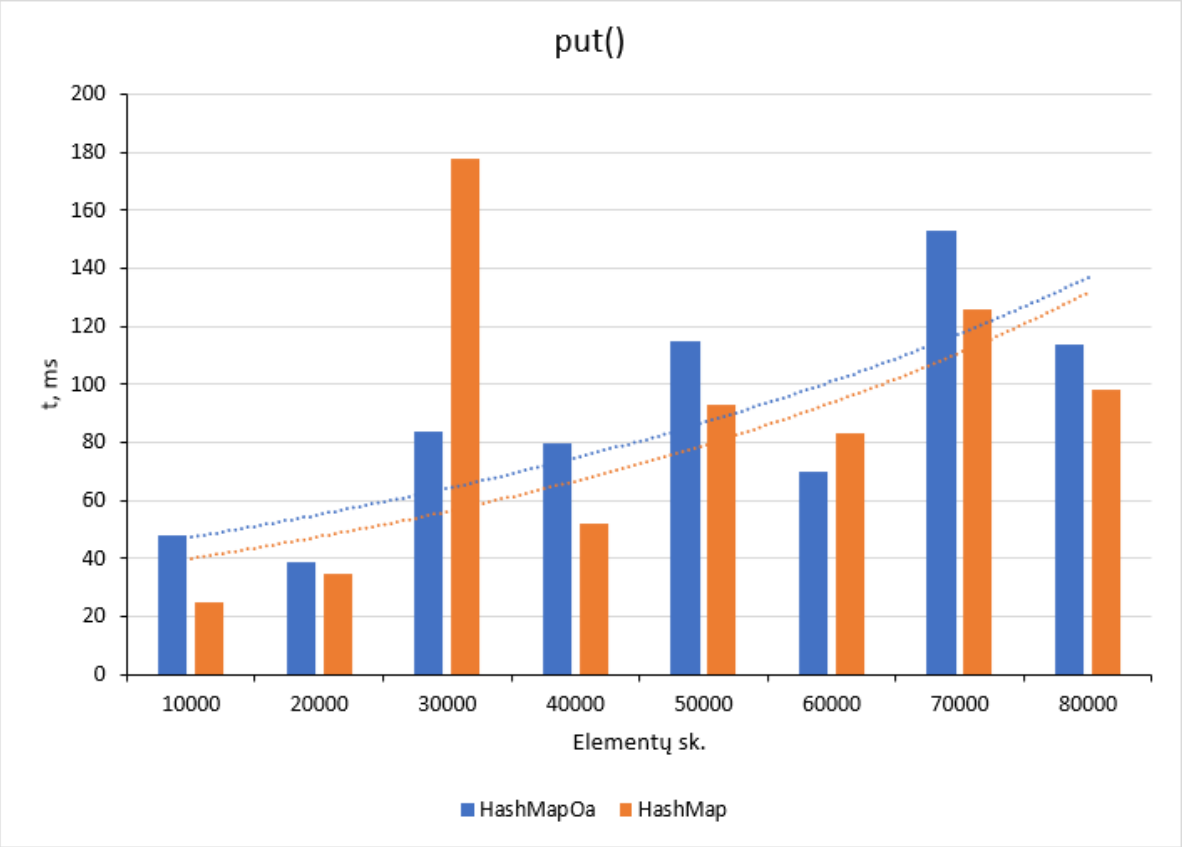
```
    @Override
    public int size() {
        return size;
    }

    @Override
    public void clear() {
        Arrays.fill(table, null);
        size = 0;
    }

    @Override
    public String[][] toArray() {
        String[][] result = new String[table.length][];

        for (int i = 0; i < table.length; i++) {
            result[i] = new String[]{table[i].toString()};
        }

        return result;
    }

    @Override
    public V put(K key, V value) {
        if (key == null || value == null)
            throw new IllegalArgumentException("Key or value is null in put(K key,
V value)");

        table[getUnusedIndex(key)] = new Entry<>(key, value);
        size++;

        if (size + deletedEntries > table.length * loadFactor)
            rehash();

        return value;
    }

    @Override
    public V get(K key) {
        if (key == null)
            throw new IllegalArgumentException("Key is null in get(K key)");

        if (getIndex(key) != -1)
            return table[getIndex(key)].value;
        else
            return null;
    }

    @Override
    public V remove(K key) {
        V removedValue = get(key);

        if (removedValue != null) {
            table[getIndex(key)] = new Entry<>(null, null);
            size--;
            deletedEntries++;
        }

        return removedValue;
```

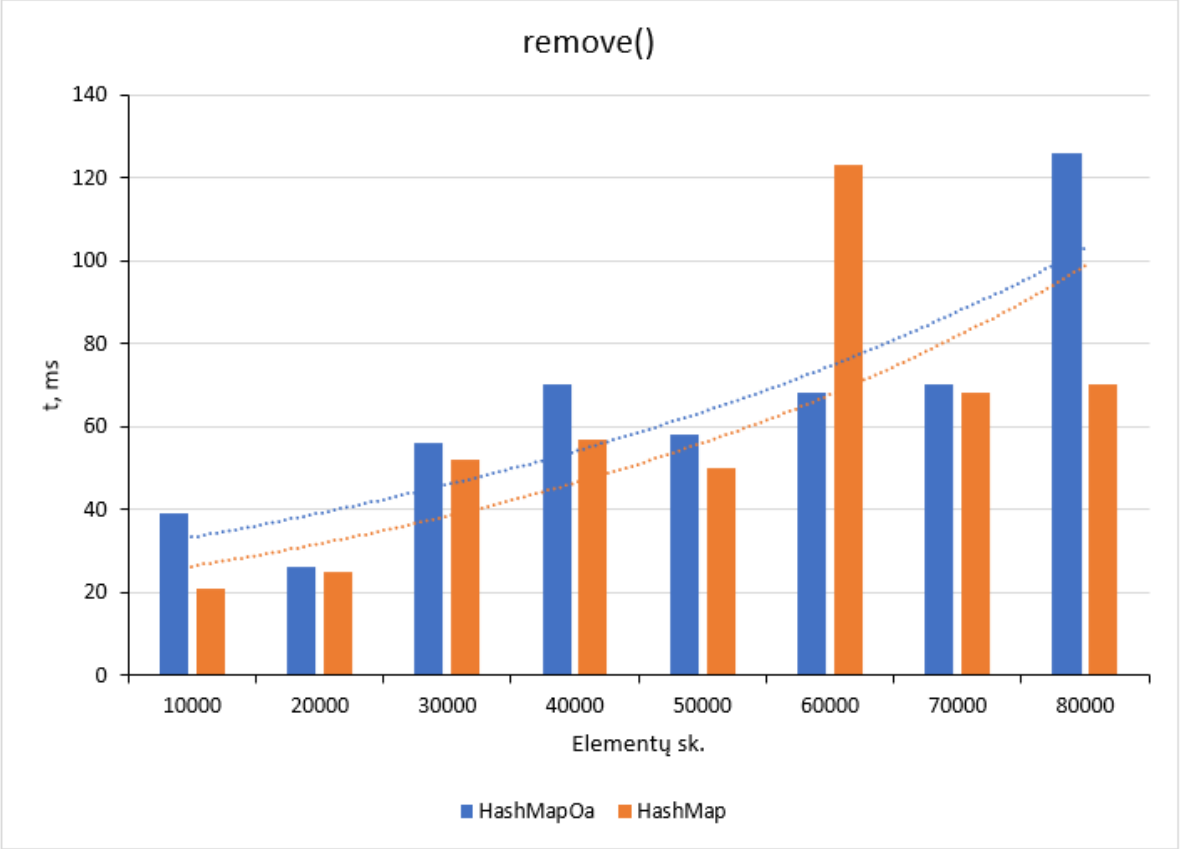
```
    }

    @Override
    public boolean contains(K key) {
        return get(key) != null;
    }

    private int getIndex(K key) {
        int index = hash(key, ht);

        for (int i = 0; i < table.length; i++) {
            int tempIndex = (index + i) % table.length;
            Entry<K, V> tempEntry = table[tempIndex];

            if (tempEntry != null && tempEntry.key != null &&
tempEntry.key.equals(key)) {
                return tempIndex;
            }
        }

        return -1;
    }

    private int getUnusedIndex(K key) {
        int index = hash(key, ht);

        for (int i = 0; i < table.length; i++) {
            int tempIndex = (index + i) % table.length;
            Entry<K, V> tempEntry = table[tempIndex];

            if (tempEntry == null || tempEntry.key == null)
                return tempIndex;
        }

        return -1;
    }

    private int hash(K key, HashType hashType) {
        int h = key.hashCode();

        switch (hashType) {
            case DIVISION:
                return Math.abs(h) % table.length;
            case MULTIPLICATION:
                double k = (Math.sqrt(5) - 1) / 2;
                return (int) (((k * Math.abs(h)) % 1) * table.length);
            case JCF7:
                h ^= (h >>> 20) ^ (h >>> 12);
                h = h ^ (h >>> 7) ^ (h >>> 4);
                return h & (table.length - 1);
            case JCF8:
                h = h ^ (h >>> 16);
                return h & (table.length - 1);
            default:
                return Math.abs(h) % table.length;
        }
    }

    private void rehash() {
```

```
        HashMapOa<K, V> newMap = new HashMapOa<>(table.length * 2, loadFactor,
ht);

        for (Entry<K, V> i : table)
            if (i != null && i.key != null)
                newMap.put(i.key, i.value);

        table = newMap.table;
        deletedEntries = 0;
    }

    private static class Entry<K, V> {

        protected K key;
        protected V value;

        protected Entry() {

        }

        protected Entry(K key, V value) {
            this.key = key;
            this.value = value;
        }

        @Override
        public String toString() {
            return key + "=" + value;
        }
    }
}
```

Greitaveika

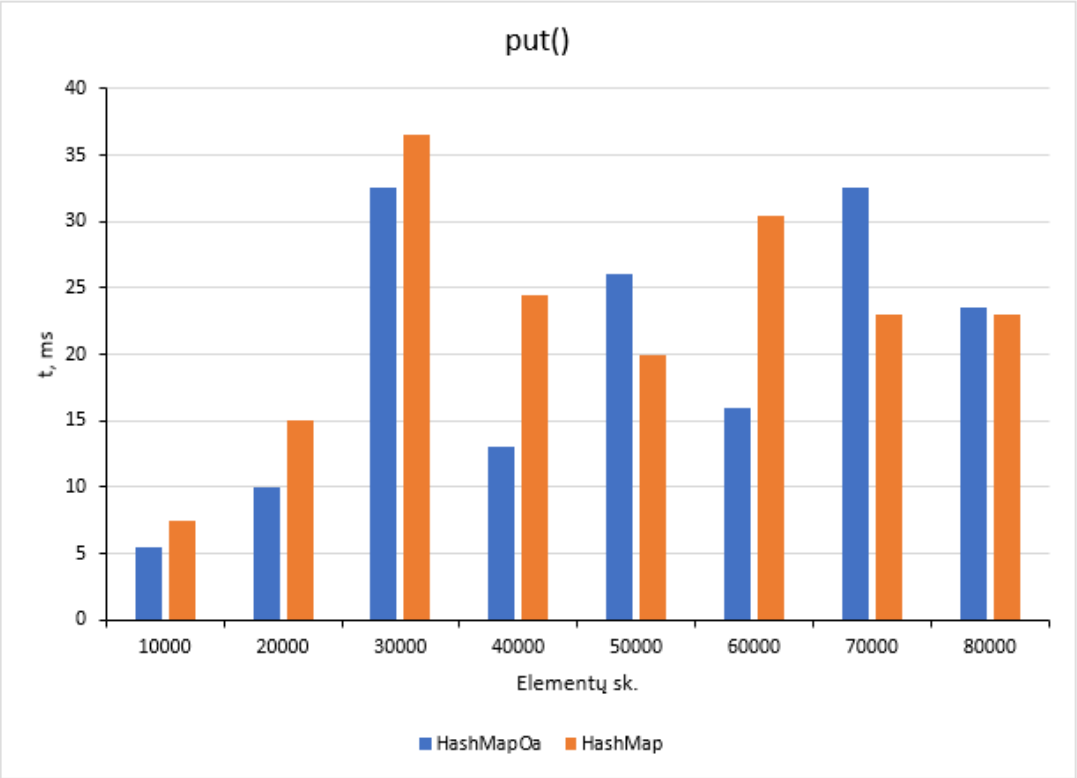
```
Dydis - 10000                    Dydis - 50000
HashMapOa put()   48 ms          HashMapOa put()  115 ms
HashMap   put()   25 ms          HashMap   put()   93 ms

Dydis - 10000                    Dydis - 50000
HashMapOa remove()   39 ms       HashMapOa remove()   58 ms
HashMap   remove()   21 ms       HashMap   remove()   50 ms

Dydis - 20000                    Dydis - 60000
HashMapOa put()   39 ms          HashMapOa put()   70 ms
HashMap   put()   35 ms          HashMap   put()   83 ms

Dydis - 20000                    Dydis - 60000
HashMapOa remove()   26 ms       HashMapOa remove()   68 ms
HashMap   remove()   25 ms       HashMap   remove()  123 ms

Dydis - 30000                    Dydis - 70000
HashMapOa put()   84 ms          HashMapOa put()  153 ms
HashMap   put()  178 ms          HashMap   put()  126 ms

Dydis - 30000                    Dydis - 70000
HashMapOa remove()   56 ms       HashMapOa remove()   70 ms
HashMap   remove()   52 ms       HashMap   remove()   68 ms

Dydis - 40000                    Dydis - 80000
HashMapOa put()   80 ms          HashMapOa put()  114 ms
HashMap   put()   52 ms          HashMap   put()   98 ms

Dydis - 40000                    Dydis - 80000
HashMapOa remove()   70 ms       HashMapOa remove()  126 ms
HashMap   remove()   57 ms       HashMap   remove()   70 ms
```

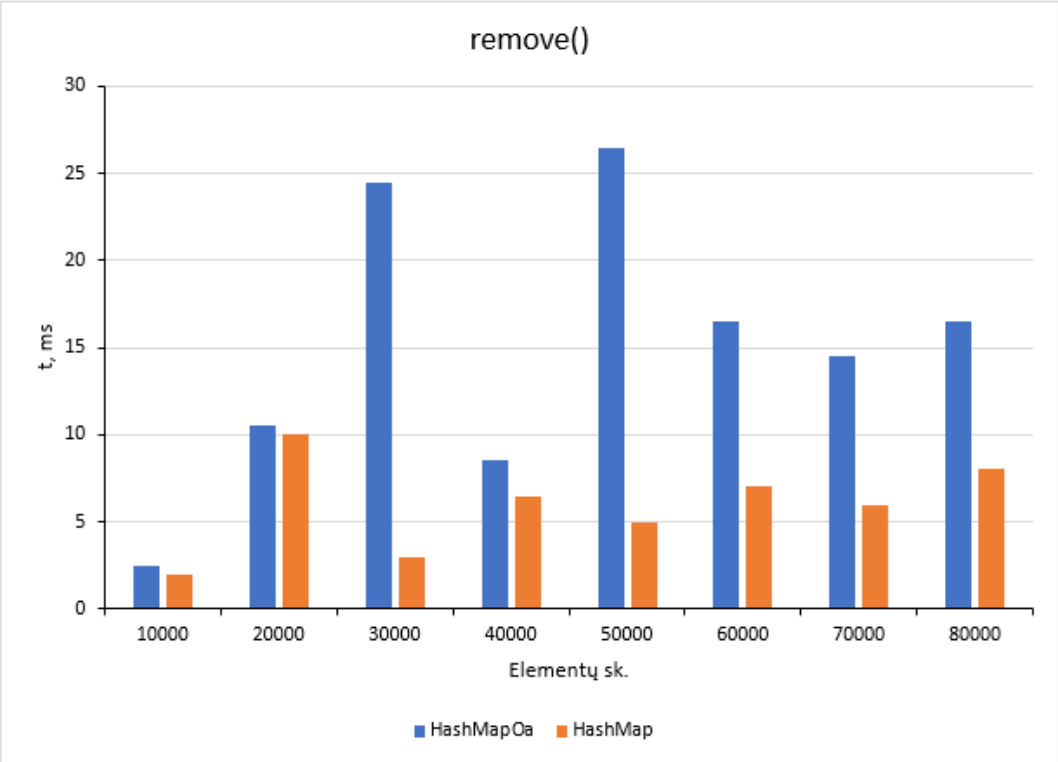




Matome, kad ir įdėjimo, ir pašalinimo metodai vidutiniškai užtrunka ilgiau naudojant HashMapOa nei HashMap. Skaičiuojant sugaištą laiką atliekant veiksmus pasitaikė kelios

reikšmės, kurios stipriai nukrypusios nuo vidurkio. Tai galėjo nutikti, nes įrašinėjant (ir ištrinant) reikšmes, jos buvo skaitomos iš failo. Pamėginau surašyti visus žodžius į masyvą prieš atliekant pridėjimą ir šalinimą iš HashMap'ų ir gavau tokius rezultatus:





Matome, kad vis tiek išlieka kelios reikšmės, kurios yra nukrypusios nuo vidutinės reikšmės. Iš antrų grafikų matome, kad didėjant elementų kiekiui, atsiranda tendencija ilgėti operacijoms.

Tai tikriausiai atsitinka dėl to, kad didėja kolizijų kiekis ir operacijos nebeatliekamos per konstantinį laiką.

Šalinimo operacija vidutiniškai užtrunka ilgiau HashMapOa struktūroje nei HashMap, kur naudojama grandinėlės kolizijų sprendimo būdas. Tuo tarpu pridėjimo operacija atliekama greičiau HashMapOa struktūroje, tačiau skirtumas yra ganėtinai smulkus (<5 ms), tad tikėtina, kad atsirado kitų veiksnių, lėtinančių veikimą.